ECool Corrector Voltages

Local application

Fri, Dec 9, 2005

ECool supports about 256 corrector power supplies whose currents are already being monitored by the control system. From time to time, there have been found to be shorts that do not cause the measured currents to change, since the supplies are current regulated. An effort to monitor the voltages of these supplies should reveal such shorts. The interface to the embedded controller is done via ethernet using a UDP-based protocol called CEC (Compact Ethernet Communication) that was designed for the purpose by Greg Saewert and Brian Kramper. This note describes a local application called `CECV` that can access these voltages and do the required processing.

Since the voltages may vary as the power supply regulates the current, it is useful to monitor the resistance (voltage / current) instead. So in addition to using CEC to retrieve the voltage data, the LA should compute the resistances of each supply, then letting the normal alarm system compare them against the expected values for each.

The LA periodically sends a request message (10 bytes) to ask for the voltage reading of up to 128 power supplies. (There are two banks of 128 supplies.) The reply to this request is a 10-byte header followed by an array of data words, each of which has the raw data format:

Raw value	*Voltage reading*
`0xFFFF`	+10.24
`0x8000`	0
`0x0000`	–10.24

Turn these values into a ordinary integers for computation by toggling the high bit.

LA parameters for `CECV`:

Prompt	*Size*	*Meaning*
`ENABLE B`	2	Usual enable Bit#
`PERIOD`	2	Data collection period in 15 Hz cycles
`MSG TYPE`	2	Message type# in CEC protocol
`FIRST`	2	Initial element in request
`TARG IP`	4	Embedded controller IP address
`RESULT C`	2	Target Chan# for receiving resistance readings
`NCHANS`	2	Number channels
`CURRENT C`	2	Local Chan# base for currents

The period must allow time for the reply data to be returned. If the reply is not returned within the specified period, it is considered missing. There should be some means of indicating that such failures in communication are occurring. It may be useful to measure the time from request to reply as an aid in setting a suitable period.

The message type# is 0 for analog readings, but 2 is defined to be used for status words.

The initial element is probably 0. The number of elements to specify in the request message is the same as the number of channels parameter. The result channel# specifies where the computed resistance values will be stored. This will allow monitoring them via the usual alarm scan support.

The two nodes that support 128 corrector power supplies each are `058A` and `058C`. The full scale for the current readings are 1 amp, except for some that use 2 amps. The current device names end in "`I`". To define devices for the resistances, we could use names ending in "`R`". All corrector power supply voltages use the same scale factors. We can then use integer arithmetic to compute the resistance, storing it as an integer result. (An alternative is to store the results into raw floating point channels.)

The order of the retrieved voltages may not match the order of the current readings, in which case the LA must be able to find, for any voltage value, the correct current value to use to get the proper resistance value.

The computation of resistance values can use integer arithmetic. For any analog channel, two constants are used for deriving the engineering units via this simple linear formula:

```
eng = raw * fs / 32768 + off,
```

Where `raw` is the 16-bit data word, `fs` is the full scale, and `off` is the offset. Assume for this program that the offset is always zero. The current channel fullscale can be `Ifs`, either 1 or 2 amperes, and the voltage fullscale can be `Vfs`, which is 10.24 volts. Let the raw readings be `Iraw` and `Vraw`. So we have:

```
V = Vraw * Vfs / 32768
I = Iraw * Ifs / 32768
R = Rraw * Rfs / 32768

R = V / I
```

Note that `Vfs` = 10.24, and `Ifs` may be 1 or 2.

Assume that `Rfs` = 10.24, meaning that we can get resistance results of ±10 ohms. (Of course, resistance should always be positive, but if `Iraw` is near zero….)

What we really want is

```
Rraw = R * 32768 / Rfs = (Vraw * Vfs) / (Iraw * Ifs) * 32768 / Rfs
```

For the case of `Ifs` = 1, assume that `Rfs` = 10.24, so we have `Rraw = Vraw * 32768 / Iraw`

For the case of `Ifs` = 2, assume `Rfs` = 5.12, so we also have `Rraw = Vraw * 32768 / Iraw`

This scheme allows using the same formula to derive `Rraw` for both cases. But it limits the range of resistance for the `Ifs` = 2 case to barely more than 5 ohms. If this is not adequate, so that we need to preserve a 10 ohm range for the result, then use `Rfs` = 20.48 and 10.24. This would change the formula to `Rraw = Vraw * 16384 / Iraw`.

To avoid divide overflow, we must have `Vraw < Iraw` for the first formulation (`Rfs` = 10.24, 5.12), but we must have `Vraw < (Iraw * 2)` for the second (`Rfs` = 20.48, 10.24).